Wissenschaftliches Arbeiten mit Python

Carsten Knoll

Technische Universität Dresden

08. November 2008

Gliederung

Python?!

EN.WIKIPEDIA:

Python is a general-purpose, high-level programming language. Its design philosophy emphasizes programmer productivity and code readability. Python's core syntax and semantics are minimalist, while the standard library is large and comprehensive. ...

- Relativ junge Programmiersprache
  (Erste Veröffentlichung: 1991, Guido van Rossum;

  Seitdem enorme Zunahme in Popularität und Entwickleraktivität)
- Schwerpunkte: Produktivität und Lesbarkeit

Warum Python in Wissenschaft nutzen? I

- Python selbst ist *quelloffen* und *frei*.
- $\exists$ sehr viele quelloffene und freie Erweiterungen.
- $\longrightarrow$ Sehr gute Vorausetzung für Forschung und Lehre (Nachvollziebar- und Verfügbarkeit, Kosten)
- Weitere Gründe:
  - Lesbarkeit
  - Produktivität
  - Generalität und Mächtigkeit
  - Plattformunabhängigkeit
  - Verbreitung (Erfahrungsaustausch, Speziallösungen)

Warum Python in Wissenschaft nutzen? II

- Eignung für Lehre bestätigt durch Veröffentlichungen und Erfahrungsberichte
- „A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering“ [7]:
  >>A corresponding solution in MATLAB is far less elegant, and it requires significant effort to achieve this in C.<<
- „Exploiting Real-Time 3d Visualisation to Enthuse Students: A Case Study of Using Visual Python in Engineering“[8]
- „Python Scripting for Computational Science“[12]
- ...

Persönliche Motivation

- Pythons Möglichkeiten stückweise entdeckt
  Beginn: Workshops von.. Thomas Güttler [1] und Arnd Bäcker [2]
  Von da ab Python zunehmend selbst genutzt → immer mehr Möglichkeiten entdeckt
- Bedürfnis Wissen weiterzugeben (und eigene Erkenntnisse zu ordnen)
- Heimliche Hoffnung: Vergrößerte Nutzer-Gemeinschaft leistet potentiell größeren Beitrag und ermöglicht besseren Erfahrungsaustausch
- → Vernetzung → Website [42]

Merkmale I

- Überschaubare und zum Teil selbsterklärende Syntax.
  „Python liest sich fast wie Pseudo-code.“
- Einrückungen haben semantische Bedeutung.
  $\longrightarrow$ Lesbarkeit erzwungen.
- Mächtige eingebaute Datentypen. (`list`, `dict`, ...)
- Docstrings (Vom Programm aus zugängliche Kurzdokumentation des Programmteils)
- Konzept: „Batterien inklusive“.
  $\longrightarrow$ Sehr umfangreiche Standard-Bibliothek
  (> 280 Module: ... `anydb`, ... `zipfile`, ...).

Merkmale II

- Konzept: Alles ist ein Objekt!
- Trotzdem: Nutzung von verschiedenen Paradigmen (prozedural, objektorientiert, funktional)
- Organisation von umfangreicheren Projekten durch Module, Pakete und Namensräume
- Weitreichende Möglichkeiten zur Fehlervermeidung und -suche: Exceptions, Tracebacks, integrierter Debugger, Log-Modul
- Möglichkeit von interaktiven Sitzungen →**IPython**

Python als „Taschenrechner“

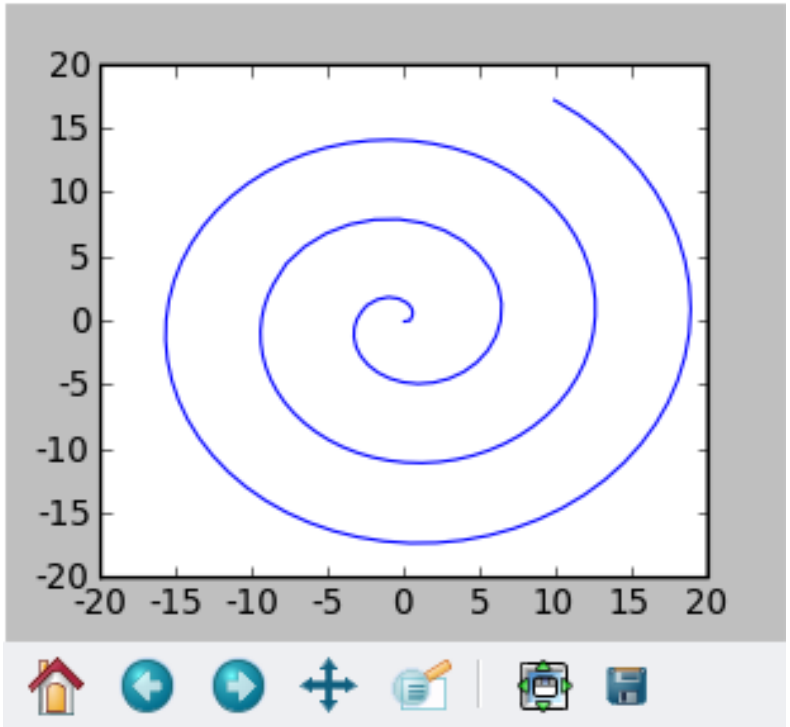
- `python` startet den Interpreter im interaktiven Modus
- selbstverständlich: +, -, *, /
  ⚠ Unterscheidung zwischen Gleitkomma- u. Ganzzahlen bei Division
- Potenzieren: `z=5**2` | `25**0.5`
- Komplexe Zahlen: `z=1+2j` | `x=z.real` | `abs(z.conjugate)`
- Für weitere Bedürfnisse: `from math import *`
  → `sin`, `atan2`, `exp`,...

Python als „Taschenrechner“ II

... bzw. gleich `ipyton -pylab` nutzen

- Unterstützung für Arrays und Matritzen `dot`, `cross`, `eigenvalues`
- Visualisierung über *matplotlib*

```
t=arange(0,20,0.1)
z=t*exp(1j*t)
plot(z.real, z.imag)
```



IPython - Verbesserte Python Shell

- Zwischenschicht zwischen Benutzer und Python-Interpreter, selbst in Python implementiert
- Zahlreiche Vorteile gegenüber ursprünglicher Shell:
  - Auto-Vervollständigung (auch auf Objektebene)
  - Farbige Ausgabe, ausführliche Ausgabe
  - „Magische Kommandos“ für schnellen Zugriff auf Docstrings, auf Quellcode, auf Betriebssystembefehle ...
  - ... log-Funktion, timing-Funktionen
  - intelligente Ein/Ausgabe-Historie
- Kurzer Überblick über Möglichkeiten: Befehl: ?
- Weiterer Einsatzmöglichkeit: als *eingebettete Shell* in eingenem Python Skript
- Sehr geeignet, um Fehler zu finden, Verhalten von Objekten zu testen, usw.

Module und Pakete

- Modularisierung: immer wichtiger mit zunehmender Komplexität
- Hauptgründe: Quellcode-Wartung und -Wiederverwertung, Gruppierung von Funktionalität (z. Bsp. für Veröffentlichung)
- In einem Modul können definiert sein: Funktionen, Klassen, Variablen
- Hierarchische Paketstruktur:
  - $\{\text{Modul}_1, \text{Modul}_2, \ldots\} = \text{Paket}_1$
  - $\{\text{Paket}_1, \text{Modul}_3, \ldots\} = \text{Paket}_2$
  - Modul: Datei mit Endung `.py`
  - Paket: Verzeichnis welches mindestens `__init__.py` enthält
- Einbinden mittels `import`
- Einbinden des kompletten Moduls, oder Teile davon

import von Modulen

komplett, in eigenen Namensraum

```
import math
x=math.cos(3.0/2*math.pi)
```

partiell, in globalen Namensraum

```
from math import cos, pi
x=cos(3.0/2*pi)
```

komplett, in globalen Namensraum

```
from math import *
x=cos(3.0/2*pi)
```

NUMPY

- Basispaket für numerische Berechnungen: `numpy`, [18]
- Herzstück: Klasse für n-dimensionale Datenfelder (Arrays)
- dazu: umfangreiche Unterstützungs-Funktionalität (`fft`, `linalg`, `random`, ...)
- Ziel: Performanz
- Bietet gute Schnittstellen zu kompiliertem Code (C und Fortran)

- Vorgänger `Numeric` und `numarray` sind in `numpy` aufgegangen; werden nicht weiter entwickelt
- Aktuell: `numpy` $\in$ `scipy`-Projekt

SCIPY

- Aufsatz auf `numpy`: scipy, [36]
- Bietet sehr viel Funktionalität aus dem Bereich wissenschaftliches Rechnen:
  - Optimierung
  - Statistik
  - Signalverarbeitung
  - DGL-Integratoren (ODE-Solver)
  - ...
- Vordefinierte spezielle Funktionen (Besselfktn., ellipt. Integrale, ...)
- Schnittstellen zum Daten-In/-Export (bspw. aus Bildern, Datenbanken)

ScientificPython

- Weitere Bibliothek für Berechnungen: ScientificPython, [35]
- Ähnliche Zielstellung zu `scipy`, trotzdem Unterschiede
- Bietet:
  - Geometrie (Vektoren, Quaternionen, Tensoren,...)
  - E/A-Schnittstellen: (PDB, NetCDF,...)
  - Parallelisierung (Threading, MPI, BSP)
  - Visualisierungsschnittstellen (VRML, ...)
  - Automatisches Differenzieren
  - ...

- `ScientificPython` verwendet `Numerical` statt `numpy`
  $\rightarrow$ evtl. Konflikte/Inkonsistenzen mit `scipy`
- Mit etwas Vorsicht (Namensräume) parallele Nutzung gut möglich

SYMPY

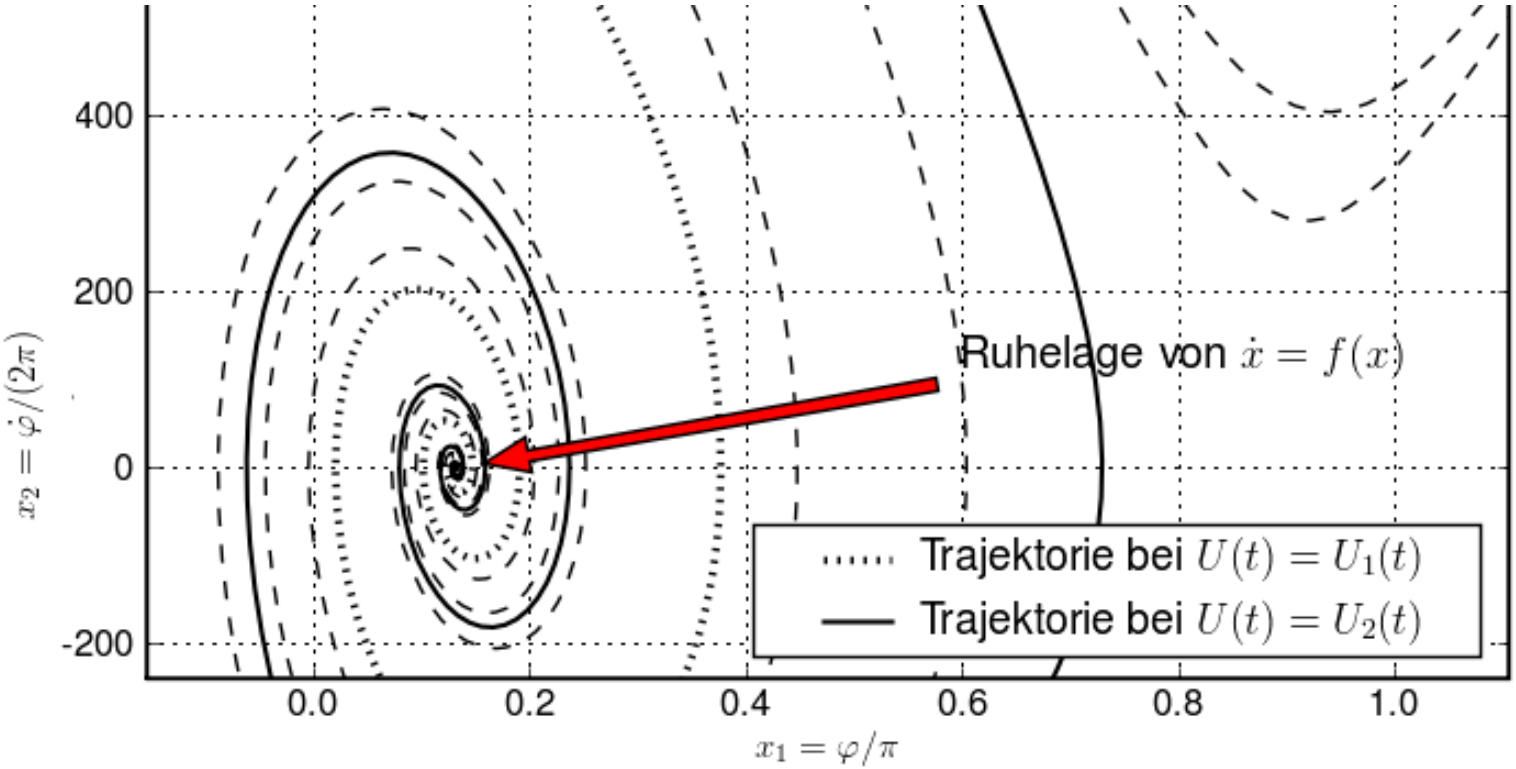
- *Symbolisches* Rechnen in Python: `sympy` [39]
- Bietet:
  - Grundlegende Rechnenoperationen
  - Termmanipulation (Zusammenfassen und Ausmultiplizieren)
  - Symbolisches Differenzieren und Integrieren
  - Grenzwertberechnung
  - Matritzen-Rechnung (symbolische Determinante, Inverse, ...)
  - Lösen von Gleichungen und Gleichungssystemen
  - Pretty-Printing und LaTeX-Ausgabe
  - 2D und 3D Visualisierung
  - ...
- Komplett in Python geschrieben
  $\longrightarrow$ mögliches Anschauungsobjekt
  $\longrightarrow$ relativ einfach anpassbar
- Ein Grund, diesen Vortrag zu halten (ich hätte es selbst gerne schon eher gekannt)

weitere Pakete

- Schnittstelle zur **G**NU **S**cientific **L**ibrary: `pyGSL`, [21]
- Swiginac, Paket für symbolische Berechnungen [38]
- Schnittstelle zu Matlab: `pyMat` [25]
- Schnittstelle zu R (freie Statistik-Programmiersprache): `rpy` [33]
- Python Aufsatz auf viele Mathe-Tools (Maple, Maxima, Octave,...): SAGE [34]
- Klimamodellierung: CliMT [5]
- Paket für Molekularbiologie: `biopython` [4]
- Geographie-Paket (Geospatial Data Abstraction Layer): GDAL [9]

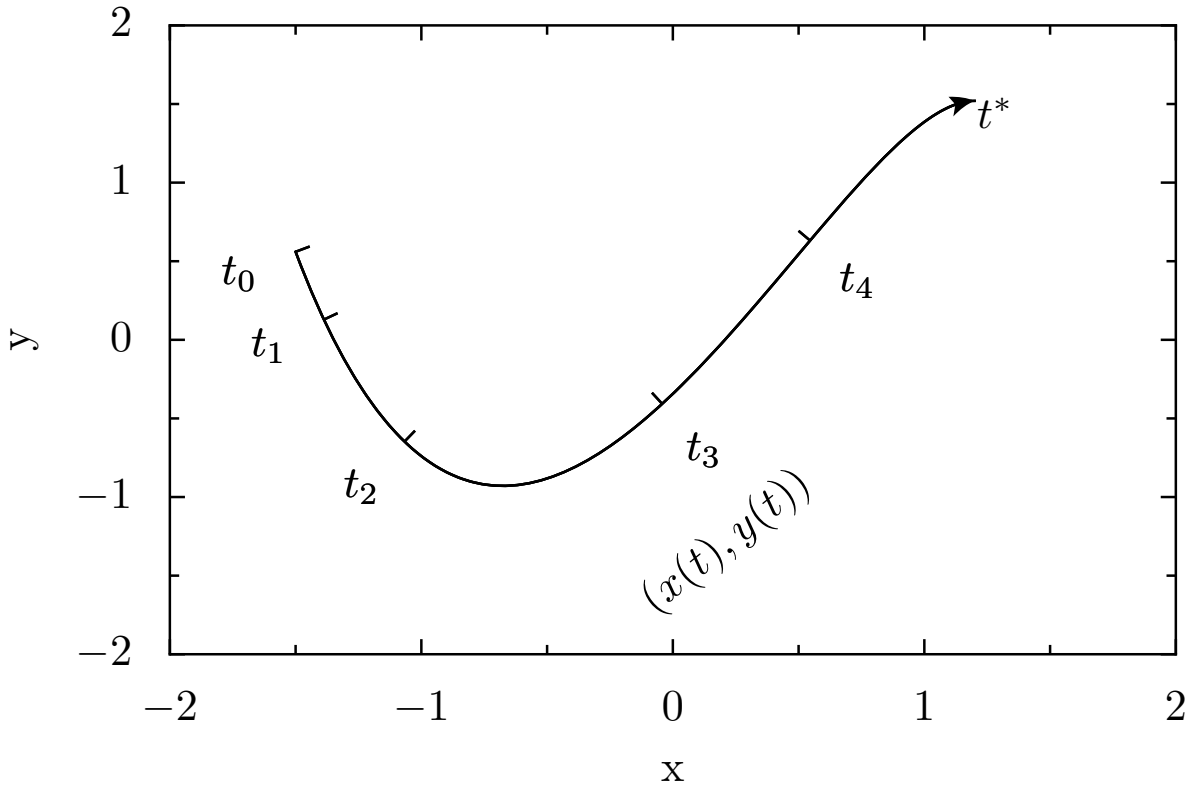
- $\exists$ noch viel mehr Pakete, mit teilweise sehr speziellen Zielgruppen
- Anlaufpunkte für Recherche: PyPI [27] und Web-Site zu diesem Vortrag [42]

- matplotlib [14]: Paket zur 2D-Visualisierung
- Orientiert sich stark an Matlabs Plot-Funktionen
- Interaktive Anzeige, mit innovativer stufenloser Skalierungs-Funktion
- LaTeX-Beschriftung an Achsen und in Legende
- Ausgabeformate: <GUI>, PNG, JPEG, PDF, SVG, PS, EPS
- Beispiel:



PYX

- pyx [30]: weiteres Visualisierungspaket (2D, 3D)
- Eigenes Zeichenmodell mit Zugriff auf alle PostScript-Fähigkeiten
- $\longrightarrow$ sehr weitreichende Funktionalität (stark objektorientiert)
- LaTeX-Beschriftungen überall
- Ausgabeformate: PDF und EPS $\longrightarrow$ nicht interaktiv
- Beispiel:

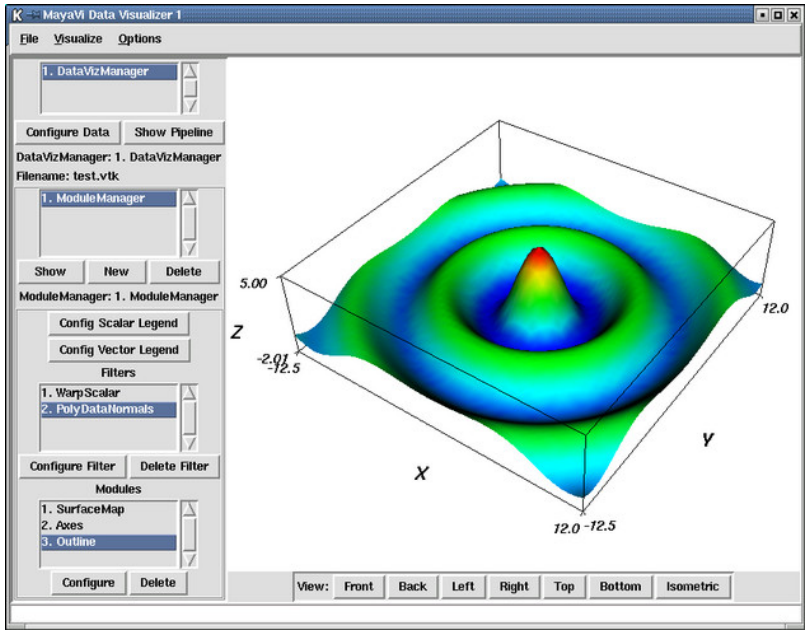


MAYAVI

- Umfangreiche 3D-Visualisierung: mayavi [15]
- Basierend auf „Visualisation Toolkit“ (C++-Visualisierungs-Bibliothek mit Python-Schnittstelle)
- Kann als eigenes Visualisierungsprogramm (wie gnuplot) gestartet werden...
- ... oder aus python-Skript heraus:

```
import mayavi
...
```

Beispiel:



PYGAME

- Eigentlich Paket für Spieleentwicklung: `pygame` [22]
- Elementare Grafikausgabe vía SDL
- Gut geeignet für Simulationen mit Echtzeit-Visualisierung
- Kommunikation mit Benutzer einfach zu implementieren
- Interessantes Konzept der Online-Dokumentation: Kommentarmöglichkeit zu jeder einzelnen Methode, Link zur Code-Suche

Weitere Visualisierungstools

- PyNGL [29]: 2D Visualisierung
- VPython [41] Echtzeit 3D Visualisierung von Objekten
- Gnuplot.py [10]: Schnittstelle zu dem bekannten und bewährten Plotting-Tool Gnuplot
- ~~scipy.gplt: Gnuplot Schnittstelle von scipy~~ **veraltet**
- pygraphviz [20]: Visualisierung von Graphen (im Sinne der Graphentheorie)
- Mehr im Python-Package-Index [28]

Korananalyse

- Python als Werkzeug für Geisteswissenschaften? Für islamische Theologie??
- In einem Beitrag [11] auf dem Python-Forum ging es wirklich darum!
- bestimmte Text-Repräsentationsform in eine andere überführen
- Häufigkeiten von Wörtern und Wortformen bestimmen
- Fernziel: semantische Vernetzung des Textes

- einfacher Zugriff auf Strings, Listen und Dictionaries
  $\longrightarrow$ Python u. a. deshalb gut geeignet für Sprachverarbeitung
- $\exists$ Werkzeugsammlung und Buchveröffentlichung [3] zu diesem Thema („Natural Language Processing“)

Automatisierte Modellbildung

- Modellbildung von physikalischen Systemen über Lagrange-Formalismus:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \delta_i$$

- L ist (ggf. umfangreicher) algebraischer Ausdruck in q_i und $\dot{q}_i$
- $\rightarrow$ Berechung der partiellen Ableitungen „per Hand“ aufwendig und fehleranfällig.
- `sympy` kann das schneller und sicherer
- Skript: Ausdruck für L u. Randinformationen einlesen und vorverdauen
- Symbolische Berechnung ausführen lassen
- Ergebnis bei der Gelegenheit auch gleich noch linearisieren
- mehr dazu: Workshop

Dateisystem durchforsten

- Nahezu trivialer Zugriff auf Systemkommandos
  Bsp: `os.system('cp datei.txt datei_sicher.txt')`
- Einfaches Durchwandern des Verzeichnisbaums mit `os.walk`

- Anwendung:
  - Verschiedene Simulationsmodelle in jeweils einem Verzeichnis
  - Alle Modellverzeichnisse haben gemeinsames Wurzelverzeichnis
  - In jedem Modellverzeichnis eine Beschreibungsdatei für das konkrete Modell
  - Skript geht durch alle Modellverzeichnisse, sammelt Beschreibungen und fügt sie zu einer großen Datei zusammen
- Weitere Anwendungen davon: gezieltes Umbenennen von vielen Dateien, spezielle Suchmethoden, ...

mögliche Lernmethode

Wie am besten (schnellsten) mit unbekanntem Paket zu Ergebnis kommen?

- Zielstellung einigermaßen präzisieren
- Beispiele (Screenshots) nach möglichst ähnlicher Lösung durchsuchen
- Konsistenz der API unterstellen und interaktiv „herumprobieren“ (IPython)
- Dabei leiten lassen von: Objektnamen [TAB], Docstrings [?], Quellcode [??]
- Ggf. Referenz oder API-Doku durchsuchen
- Bei tieferem Interesse: Tutorials durcharbeiten, dann Referenz und Quellen

Fehler vermeiden...

- Fallen kennen
  - Ganzzahldivision[1]: `3/2` → `1`!
  - Einrückungen (sehr gefährlich: TABs und Leerzeichen gemischt)
  - Erstes Argument von Objektmethoden (immer Objekt selbst: `self`)
  - Nicht-ASCII Zeichen in Kommentaren oder Strings
  - Weitere: [42]

- Fallen umschiffen
  - Entweder Gleitkommadivision nutzen (`3.0/2`)
    oder `from __future__ import division`
  - Nur Leerzeichen verwenden, 4 pro Stufe, LZ von IDE anzeigen lassen
  - Merken! Oder IDE verwenden, die `self` automatisch einfügt
  - Kodierung vereinbaren: Am Dateianfang: `# -- coding: utf8 --`
  - `pylint` [24] oder ähliches benutzen; Werkzeug zur Überprüfung von Code.

[1] Änderung ab Version 2.6

... oder wenigstens schnell finden

- Sicherheitsabfragen (Typ von Objekten, Länge von Listen)
  $\longrightarrow$ helfen Fehler in der Nähe ihres Ursprungs zu finden
- Logging [43] nutzen ist besser und flexibler als ständiges Aus- und „Einkommentieren“ von `print`-Anweisungen
- Unittest (Automatisierte Überprüfung bestimmter Merkmale)
  siehe z.Bsp. `pylib` [23], oder `doctest` [6]
- Ausnahmen-Behandlung aktiv nutzen: `raise`, `try`, `except`
- Eingebettetes IPython immer importieren ...
  ($\longrightarrow$ sorgt für farbige und aussagekräftige tracebacks)
  ... und an zweifelhaften Stellen Shell aufrufen (siehe Workshop)
- integrierten Debugger kennenlernen
  (trotz erstmal umständlicher Text-Bedienung)

Probleme

- Schreibfehler fallen oft erst zur Laufzeit auf
  Grund: Kein Kompiliervorgang, keine statische Typbindung, keine explizite Deklaration (es sei denn, man nutzt `traits` [40])
- Mehrfachimplementierungen (inkompatibel)
  z.Bsp. `sympy.sin` und `scipy.sin`
- Manchmal auftretende Inkonsistenzen in Bezeichnungen und Verhalten
  Bsp:
  - `einString.upper()` gibt eine Kopie des Strings in Großbuchstaben zurück
  - `eineListe.reverse()` gibt nichts zurück, sondern dreht die Reihenfolge der Elemente von `eineListe` um
- Ausführungsgeschwindigkeit bei rechenintensiven reinen Python-Implementierungen verbesserungsfähig

---

- Python 3000 geht einige dieser Probleme an
- Und außerdem: Freie Software lebt von Partizipation!
  $\longrightarrow$ Eigeninitiative

Versionskontrolle mit Mercurial

- Warum Versionskontrolle („Revision Control“)?
  - Archivierung und Wiederherstellung
  - Protokollierung (wer, wann, was, (warum))
  - Koordinierung von mehreren Bearbeitern

  - Anwendbar auf alle Typen von Dateien (auch binäre Formate)
  - Besonders geeignet: Programmcode, LaTeX-Code, sonstige Textformate

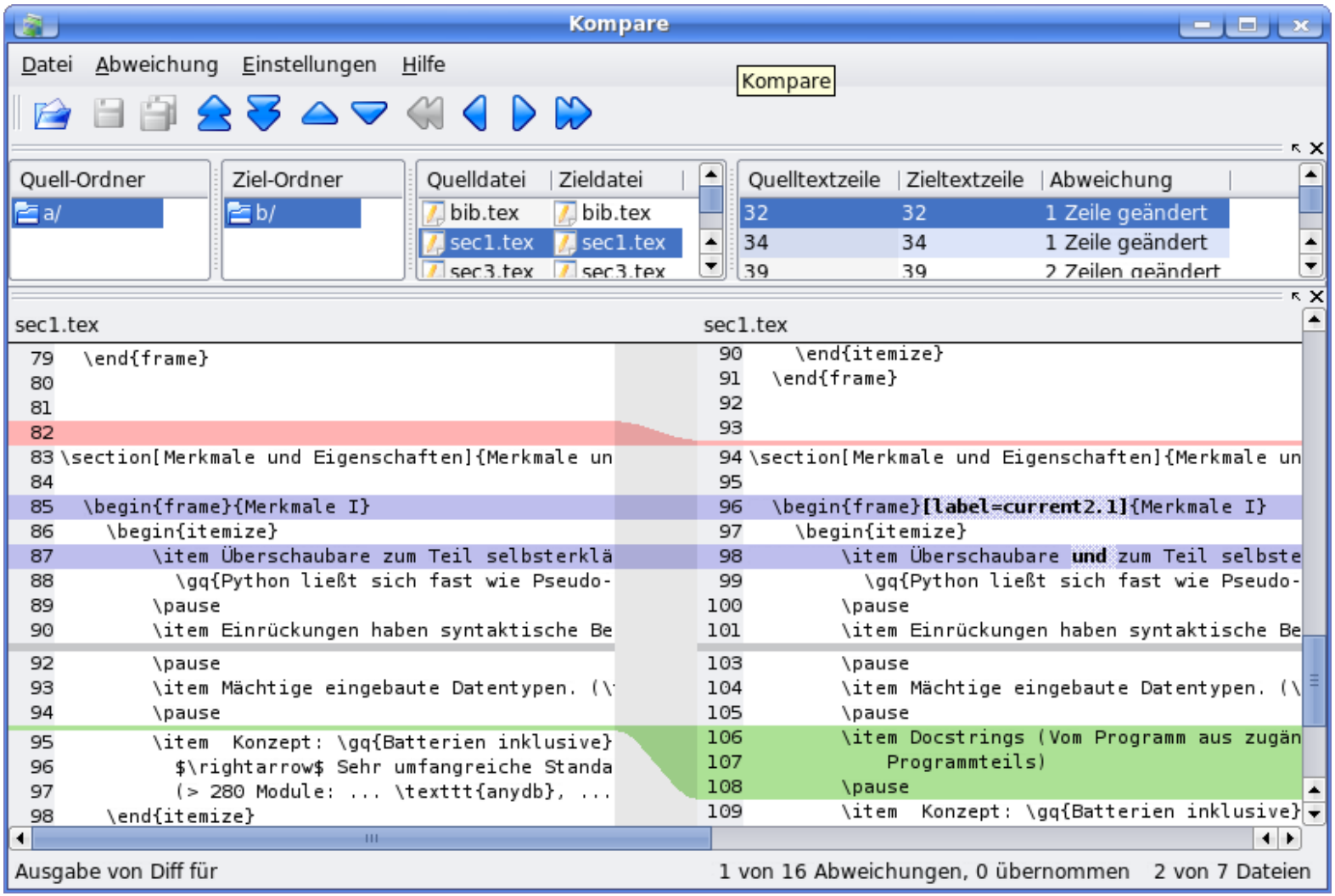
- Warum Mercurial [16] benutzen?
  - *Verteiltes* Versionskontrollsystem ($\neq$ SVN, CVS)
  - $\rightarrow$ Auch gut für lokale (1-Benutzer-) Anwendung geeignet
  - Frei, quelloffen, in Python (und C) implementiert, plattformunabhängig

  ---

  - Persönlich bisher nur Mercurial ausprobiert ;)
    $\rightarrow$ Positive Erfahrungen

- Anwendung im Workshop

- Mein Lieblings-Merkmal: Ausgabe von `hg diff` zu graphischem Diff-Programm umlenken, z. Bsp: `hg diff | kompare -`



sonstige Tipps

- Immer (mindestens) eine IPython-Shell offen haben für schnelles Probieren
- Die „richtige“ Entwicklungsumgebung (IDE) finden: Eric, eclipse, Emacs, ...
- Das Rad nicht (immer) neu erfinden
  - Frage I: Ist mein Problem so exotisch, dass es noch nicht gelöst wurde?
  - Frage II: Was ist besser (Zeit, Lerneffekt) Lösung suchen und anpassen, oder selber lösen?
- Bei Gelegenheit mal die „offiziellen“ Konventionen für guten Python-Quellcode (PEP 8) [19] lesen
- An geeigneter Stelle: bewusste Emanzipation von Hinweisen, Konventionen, ... auch von meinen ;)

Zusammenfassung

Danke

Vielen Dank.

CarstenKnoll@gmx.de

`http://code.google.com/p/wiapy/` (bisher kein Inhalt)

Referenzen I

[1] Güttler, Thomas
Python, Programmieren macht Spaß.
http://www.thomas-guettler.de/vortraege/python/einfuehrung.html

[2]
Bäcker, Arnd
Wissenschaftliches Rechnen mit Python, Vortrag auf dem LiT 2004.
http://www.physik.tu-dresden.de/~baecker/python/linuxtag2004/linuxtag.html

[3]
Natural Language Processing in Python
Steven Bird, Ewan Klein, and Edward Loper
http://nltk.sourceforge.net/index.php/Book

Referenzen II

[4]

biopython Website
http://www.biopython.org

[5]

CliMT Website
http://maths.ucd.ie/~rca/climt/index.html

[6]

doctest Dokumentations-Website
http://www.python.org/doc/2.5.2/lib/module-doctest.html

[7]

Fangohr, Hans
A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering, M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 1210–1217, 2004.

Referenzen III

[8]

Fangohr, Hans

Exploiting Real-Time 3d Visualisation to Enthuse Students: A Case Study of Using Visual Python in Engineering, V.N. Alexandrov et al. (Eds.): ICCS 2006, Part II, LNCS 3992, pp. 139–146, 2006.

[9]

GDAL Website.

http://trac.osgeo.org/gdal/wiki/GdalOgrInPython

[10]

Gnuplot.py Website.

http://gnuplot-py.sourceforge.net/

[11]

Der Koran als Topic Map Beitrag im deutschen Python-Forum.

http://www.python-forum.de/topic-13946.html

Referenzen IV

[12]
Langtangen, Hans Petter
Python Scripting for Computational Science, Texts in Computational Science and Engineering, Third Edition Springer Heidelberg 2008

[13]
Langtangen, Hans Petter
Website zum Buch [12],
http://vefur.simula.no/~hpl/scripting/

[14]
matplotlib Website.
http://matplotlib.sourceforge.net/

[15]
mayavi Website.
http://mayavi.sourceforge.net/

Referenzen V

[16]
mercurial Website.
http://www.selenic.com/mercurial/wiki/

[17]
numpy for Matlab Users.
http://www.scipy.org/NumPy_for_Matlab_Users

[18]
numpy Website.
http://numpy.scipy.org/

[19]
PEP 8: Style Guide for Python Code.
http://www.python.org/dev/peps/pep-0008/

Referenzen VI

[20]
pygraphviz Website.
https://networkx.lanl.gov/pygraphviz/

[21]
pyGSL Website.
http://pygsl.sourceforge.net/

[22]
pygame Website.
http://www.pygame.org/

[23]
pylib Website.
http://codespeak.net/py/dist/

Referenzen VII

[24]
pylint Website.
http://www.logilab.org/857

[25]
pyMat Website.
http://claymore.engineer.gvsu.edu/~steriana/Python/pymat.html

[26]
pyparallel Website.
http://pyserial.wiki.sourceforge.net/pyParallel

[27]
Python Package Index Kategorie : Scientific/Engineering.
http://pypi.python.org/pypi?:action=browse&show=all&c=385

Referenzen VIII

[28]
Python Package Index Kategorie : Scientific/Engineering:Visualisierung.
http://pypi.python.org/pypi?:action=browse&show=all&c=399

[29]
PyNGL Website.
http://www.pyngl.ucar.edu/index.shtml

[30]
pyx Website.
http://pyx.sourceforge.net/

[31]
pyxGraph Website.
http://www.physik.tu-dresden.de/~baecker/python/pyxgraph.html

Referenzen IX

[32]
pyxPlot Website.
http://www.pyxplot.org.uk/

[33]
rpy Website.
http://rpy.sourceforge.net/index.html

[34]
Sage Website.
http://www.sagemath.org

[35]
ScientificPython Website.
http://dirac.cnrs-orleans.fr/plone/software/scientificpython/

Referenzen X

[36]
scipy Website.
http://www.scipy.org/

[37]
simplyDraw Website.
http://arminstraub.com/browse.php?page=programs_simplydraw

[38]
swiginac Website.
http://swiginac.berlios.de/

[39]
sympy Website.
http://code.google.com/p/sympy/

Referenzen XI

[40]
Traits Website (Expilzite Variablen-Deklaration für Python).
http://code.enthought.com/projects/traits/

[41]
Vpython Website.
http://www.vpython.org/

[42]
wiapy Website.
http://code.google.com/p/wiapy/

[43]
The python logging module is much better than print statements, Blog von Matt Wilson
http://blog.tplus1.com/index.php/2007/09/28/the-python-logging-module-is-much-better-than-print-statem